Exhibit 2

Copy of an Internal, Confidential, and Non-Publicly Disclosed Document Written by Inventor Ikko Fushiki, entitled, "XsRGB and XsARGB Specs (v.0.4)"

(Dates Redacted)
(17 pages)

XsRGB and XsARGB Specs (v 0.4)

Ikko Fushiki

I. Introduction

Since the introduction of color devices for personal computers such as color monitors, color scanners, digital camera, and color printers, it became important for computer systems or applications to be able to produce consistent colors among different devices. In response to that demand, Microsoft and Apple introduced ICM and ColorSync, respectively. Non-OS venders like Kodak have their own color management systems. Each color management system has its own color reference frame.

Each device records the color data in its device specific manners. It is necessary to be able to interpret the device specific data to the common standard so that other devices can understand it. As a device independent space, sRGB was introduced. This represents the typical RGB profile of PC monitors. If each device can translate its own device data to sRGB color space, all the other device can understand it. However, the color device venders and publishing industry are criticizing the narrow ranges of the displayable range of the color space (hereafter it is called gamut) and the low accuracy of sRGB data. The gamut of sRGB does not cover the displayable colors of all monitors, printers, or scanners. The storing the image data in 8 bit per color channel is not good enough when the color calibration or image processing is applied to the image. Indeed the recent scanner or digital cameras can take images in more than 8-bit in each color channel. We need to have a standard to be able to handle the data in higher accuracy.

Here we have introduced the new device independent RGB color space called XsRGB color space to address those issues of color range and image degradation. This XsRGB color space has the higher accuracy, and has the wider gamut than sRGB color space. It covers more than visible color space. Any existing device color reference frame in other color managements can be expressed in terms of the XsRGB color space without any loss of gamut and rounding errors.

The advanced graphics system requires the features of anti-aliases (removing ragged edges) and blending (translucency) effects. Those effects are handled by an extra component called "alpha channel" in addition to RGB components. In order to perform the anti-alias and blending operation correctly with the alpha channel, we need to have the linear color components in terms of their intensities. However, sRGB or RGB spaces in other color managements store the color values in non-linear 8 bits per channel. Its non-linearity is expressed usually as "gamma" value. The gamma values of Microsoft and Apple's color management systems are 2.2 and 1.8, respectively. This was important to keep the data in non-linear way when the each component is restricted to 8 bit. When the non-linear 8 bit data is converted to the 8 bit linear value, the humans can notice

contours in images since the humans can perceive more detailed color changes in low intensity values. However, when the size of each component is extended to higher bit (10 bit or higher), those gaps are not noticeable. We can certainly store the data linearly in 16 bit component without having noticeable contour effects.

II. Color Spaces

The light is made of electro-magnetic waves as discovered by a physicist Maxwell in 19th century. In early 20th century, Einstein has pointed out that the light displays the properties as energy particles called photons. Hence the light can be classified according to either by wave-length or by energy. The properties of light are very different according to its wave-lengths. We usually call the light radio, microwave, infra-red, visible, ultra-violet, X-ray, and gamma-ray according to its wave-lengths (in descending order). The wave lengths of visible light range from 400 nm to 700 nm.

The light in nature is continuous in spectrum. However, human cannot understand all the continuous spectrum. A human eye has three types of receptors to respond to that continuous spectrum of the light. It is unfortunate that we human cannot understand the detailed spectrum of the light. On the other hand, it is fortunate that we can create a device that can give us the illusion of the continuous spectra of the light by mixture of only three different colors. The work of Commission de l'Éclairage (CIE) in early 1930s gave the foundation for the human response to the color. We can regard CIE XYZ as the human device color space. CIE XYZ space is also mathematically well defined and it is possible to calculate XYZ value by measuring the spectrum of a sample color. Hence, CIE XYZ is the device independent color space that is closely tied to the human color perception.

The most convenient and well used color space in computer is RGB space. However, its RGB means different in different devices. In order to solve this confusion, sRGB color space was introduced. sRGB color space represents the color space of typical PC monitors. It has the explicit mathematical relationships with CIE XYZ color space. Hence, sRGB is the device independent color space that is closely related to PC monitors. When image is stored in sRGB color space, it will display reasonably well without any correction in most monitors since monitors have the similar device RGB spaces. This is an advantage of using sRGB. However, the problem arises when we try to import images from other types of devices such as scanners and digital cameras. Some of their displayable colors may go out of the gamut of sRGB color space. When images are stored in sRGB, some of the original information of the input devices is lost. Also we cannot utilize the printer's colors that go outside of the gamut of sRGB color space. As the device independent color space, we need the color space with a large gamut so that it can cover the gamut of existing devices. We have here introduced XsRGB color space to cover the gamut of all the existing devices. The color spaces of other management systems can be expressed in terms of XsRGB color space without any loss of gamut or precisions. XsRGB can express much more detailed shades of colors than sRGB.

III. Measurement of Color and Color Conversion

XsRGB color space is mathematically well defined as will be described later. Each color can be converted to XsRGB color by measuring a color patch by colorimeter or spectrometer.

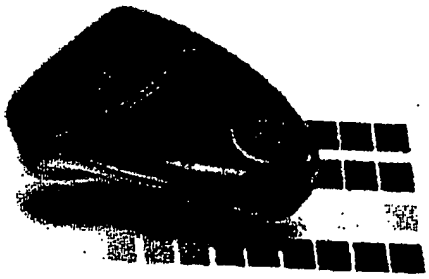

Fig. 1. *Measuring color by a colorimeter (This picture is from www.colorsavvy.com).*

When a color is measured in CIE XYZ space and is converted to XsRGB, there is no conversion error or no gamut loss. However, when CIE XYZ is converted to sRGB, there is a conversion error and also can have a gamut loss.

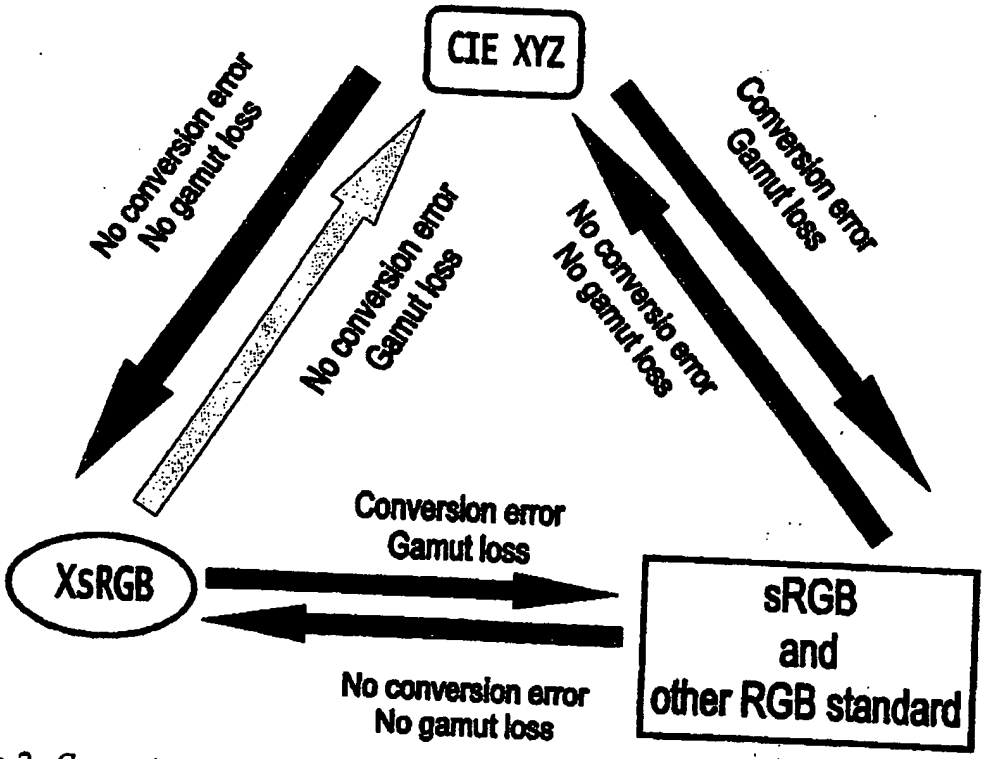

Fig. 2. *Conversion Relationships between CIE XYZ, XsRGB, and other RGB color space.*

In Fig. 2, the green arrows mean that the conversion is loss less. The red arrows mean the conversion could cause errors and gamut clipping. The yellow arrow means the converted value could go out of gamut. Since both gamuts of CIE XYZ and XsRGB are larger than visible colors. The conversion from XsRGB to CIE XYZ should not cause any visible artifacts. Other color management spaces like Apple's 13" monitor color space have the same relationships as sRGB when it is converted back and forth to CIE XYZ and XsRGB color spaces.

The difference of CIE XYZ and XsRGB is that the conversion of XsRGB to and from sRGB is very simple when a color lies in sRGB's gamut while CIE XYZ to and from sRGB needs more work. Since XsRGB itself is a RGB space, it can produce fairly good results in monitor without modification.

When a user wants to calibrate a scanner, a user first scan the picture which contains the reference patches with well known CIE XYZ values. Hence, the XsRGB values are also known. The scanned image contains the device specific RGB values for those patches. From that result, the device RGB to XsRGB conversion functions are created. The scanned images hereafter use those conversion functions to convert the device RGB to XsRGB. The scanner manufacturer should supply the default profile of the scanner.

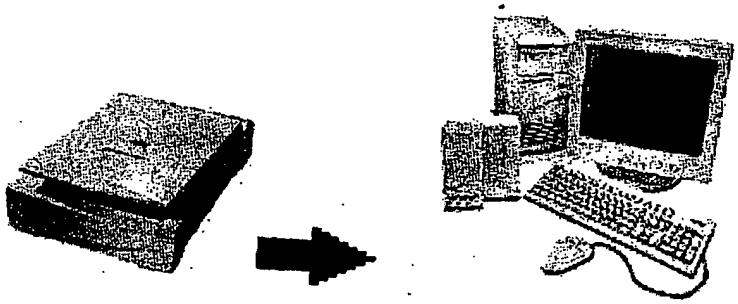

Fig. 3. *Scanning the color patches to a computer by a flatbed scanner.*

In case of calibrating a printer, a user first print a picture with XsRGB reference patches to the printer. Then the colors of the printed color patches are measured by the colorimeter. The printer profile is created so that the measured XsRGB or CIE XYZ values match with the original patches. The printer venders should supply the default profiles of the printer.

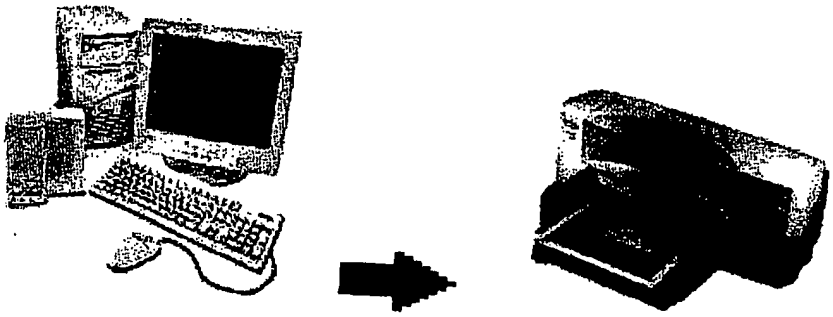

Fig. 4. *Printing the color patches to a printer.*

Usually the printer profiles are created as the white point of the 5000K Blackbody radiation (hereafter referred as D50) while sRGB has the white point of the 6500K Blackbody radiation (hereafter referred as D65). In the appearance color matches, the relative colors around the white points are matched between the printer output and monitor output. In the exact color matches, the CIE XYZ values themselves are matched in the printer output and monitor output. sRGB is defined only in D65. Hence, the appearance matches in the monitors and printers need some work. In order to correct this inconvenience we have defined XsRGB also for D50. This will ease the work of the appearance matches.

IV. Translucency and Alpha Channel

Images can be scanned, painted, or synthesized by mathematical formula. It is becoming important to be able to mix different types of images and blend them together. The advance graphics system requires the anti-aliased lines and shapes. The alpha channel is used in anti-aliasing. The alpha value is defined as the translucency of the image. 0 % means completely transparent and 100 % means completely opaque. 50 % means the background and foreground are mixed in the same intensity. In order to do blending correctly, we need to have the color values which are proportional to the luminosity. However, sRGB or other proposed RGB standard do not have the linear response. For examples, sRGB has the gamma value 2.2 and Apple's 13" monitor RGB has the gamma value of 1.8. First we must linearize those RGB values and then apply blending. When the results are stored as linear RGB with 8 bit in each channel, the 256 levels of shades is not fine enough for humans. So we tend to notice the contours in the lower colors. Because of those artifacts in 8 bit per channel, the current blending operation neglect the gamma correction. The blending operations are done as if those RGB values were linear. This creates the artifacts in the blending results. For an example, suppose one wants to blend two colors from a point 1 to a point 2. The color in the mid point does not look like the averaged color.

When the precision of each color component is increased from 8 bit to more or equal to 12 bit, humans will not notice contours even when the color values are stored linearly. In order to allow a liner color space, XsRGB requires 12 or higher bit for each channel. We can add an alpha channel in each XsRGB color component. We call this XsARGB.

V. Problem Addressed

We have addressed the following problems:

1. sRGB has narrow gamut.
2. The precision of sRGB is not accurate enough when the color calibration or image processing is applied to the image.
3. The blending and anti-alias operation needs the linear RGB.
4. The current color management is complicated.
5. The color matching is difficult between devices with different white points.
6. We need a standard format for the image archive with consistent colors without degrading the original data.
7. We need to have a bitmap format which can handle transparency and blending efficiently.
8. How do we handle the excess alpha values.
9. Sometimes, we need detailed information in dark area.

We have solutions for the above problems.

1. XsRGB allows the component of each color to be negative and to be beyond 1 (when normalized to 1 in sRGB). XsRGB's gamut is larger than visible color space. Hence, XsRGB can express any color of any device.
2. XsRGB stores each component in 16 bit or higher (possibly in floating point for 32 bit).
3. XsRGB's each component is linear (gamma = 1).
4. The color profile venders do not have to clip to the narrower gamut and they do not have to non-linearize to create XsRGB. This avoids the confusion of different gamma values in different color standards.
5. We defined XsRGB in different white points to simplify the color matching.
6. The standard images can be stored in XsRGB format without attaching ICC profile.
7. We added an alpha channel canonically to XsRGB to be able to store the information of transparency. Also RGB values can be premultiplied by alpha so that we can have the efficient blending operations.
8. We redefined the meaning of alpha channel.
9. We allow XsRGB and XsARGB to have multi-color resolutions to handle detailed information for certain range of colors.

VI. Mathematical Format of XsRGB

XsRGB is linear in intensity of each component. Hence, it can relate linearly to 1931 CIE XYZ values. Let R_0, G_0, and B_0 denote the normalized red, green, and blue components, respectively. Let X, Y, and Z denote 1931 ICE XYZ values but they are

normalized to 1 instead of 100. The relationship between the normalized XsRGB and XYZ are given by a 4x4 matrix.

$$\begin{pmatrix} R_0 \\ G_0 \\ B_0 \\ 1 \end{pmatrix} = M \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} , \tag{1a}$$

where

$$M = \begin{bmatrix} m_{RX} & m_{RY} & m_{RZ} & t_R \\ m_{GX} & m_{GY} & m_{GZ} & t_G \\ m_{BX} & m_{BY} & m_{BZ} & t_B \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1b}$$

We need only 12 coefficients to define XsRGB. In addition to the rotational part (m_{RZ}, etc.), we allow the translational part (t_R, etc.). With this notation, we can address the white point as well as black point. Using the inverse of the above matrix, the reverse relation from XsRGB to CIE XYZ space is given by

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = M^{-1} \begin{pmatrix} R_0 \\ G_0 \\ B_0 \\ 1 \end{pmatrix} , \tag{2a}$$

where

$$M^{-1} = \begin{bmatrix} n_{XR} & n_{XG} & n_{XB} & u_X \\ n_{YR} & n_{YG} & n_{YB} & u_Y \\ n_{ZR} & n_{ZG} & n_{ZB} & u_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2b}$$

We want XsRGB to have a simple transform to sRGB in D65. Indeed, we want XsRGB to be identical to sRGB when its value is inside the range of sRGB. From sRGB spec, the coefficients of Eq. (1b) and Eq. (2b) are determined as

$$M_{D65} = \begin{bmatrix} 3.2410 & -1.5374 & -0.4986 & 0 \\ -0.9692 & 1.8760 & 0.0416 & 0 \\ 0.0556 & -0.2040 & 1.0570 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3a}$$

and

$$M_{D65}^{-1} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 & 0 \\ 0.2126 & 0.7152 & 0.0722 & 0 \\ 0.0193 & 0.1192 & 0.9505 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3b}$$

The white point of D65 is $(x_{D65}, y_{D65}) = (0.3127, 0.3291)$, the corresponding CIE XYZ values are

$$\begin{cases} X_{D65} = x_{D65} / y_{D65} = 0.9502 \\ Y_{D65} = 1.0 \\ Z_{D65} = (1.0 - x_{D65} - y_{D65}) / y_{D65} = 1.0887 \end{cases} \tag{4}$$

Note that the Y-value at the white point is 1. When the device has the different white point (X_w, Y_w, Z_w), the CIE XYZ coordinates for the appearance match must be transformed by the scaling matrix

$$S_w = \begin{bmatrix} X_{D65} / X_w & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & Z_{D65} / Z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5a}$$

and its inverse is

$$S_w^{-1} = \begin{bmatrix} X_w / X_{D65} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & Z_w / Z_{D65} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5b}$$

The transformation matrix from XYZ to XsRGB at this white point is given by

$$M_w = M_{D65} S_w \tag{6a}$$

and its inverse matrix is given by

$$M_w^{-1} = S_w^{-1} M_{D65}^{-1} \tag{6b}$$

For an example, the white point of D50 is $(x_{D50}, y_{D50}) = (0.3457, 0.3585)$. The corresponding CIE XYZ value is $(X_{D50}, Y_{D50}, Z_{D50}) = (0.9643, 1, 0.8251)$. Hence the scaling matrices are

$$S_{D50} = \begin{bmatrix} 0.9854 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.3195 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7a}$$

and

$$S_{D50}^{-1} = \begin{bmatrix} 1.0148 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0.7579 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7b}$$

The resultant transformation matrices for D50 are:

$$M_{D50} = \begin{bmatrix} 3.1937 & -1.5374 & -0.6579 & 0 \\ -0.9550 & 1.8760 & 0.0549 & 0 \\ 0.0548 & -0.2040 & 1.3947 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8a}$$

and

$$M_{D50}^{-1} = \begin{bmatrix} 0.4185 & 0.3629 & 0.1832 & 0 \\ 0.2126 & 0.7152 & 0.0722 & 0 \\ 0.0146 & 0.0903 & 0.7204 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8b}$$

We can obtain the appearance match if the XsRGB values are calculated from the conversion matrix of the device white point. The absolute match can be done if the conversion matrix of D65 is used irrespective of the device white point.

Fig. 5 shows the XsRGB values at the corner of CIE xy-chromaticity diagram. From Eqs. (1a) ~ (3b), we can obtain the relationship between CIE XYZ and the normalized XsRGB components in D65.

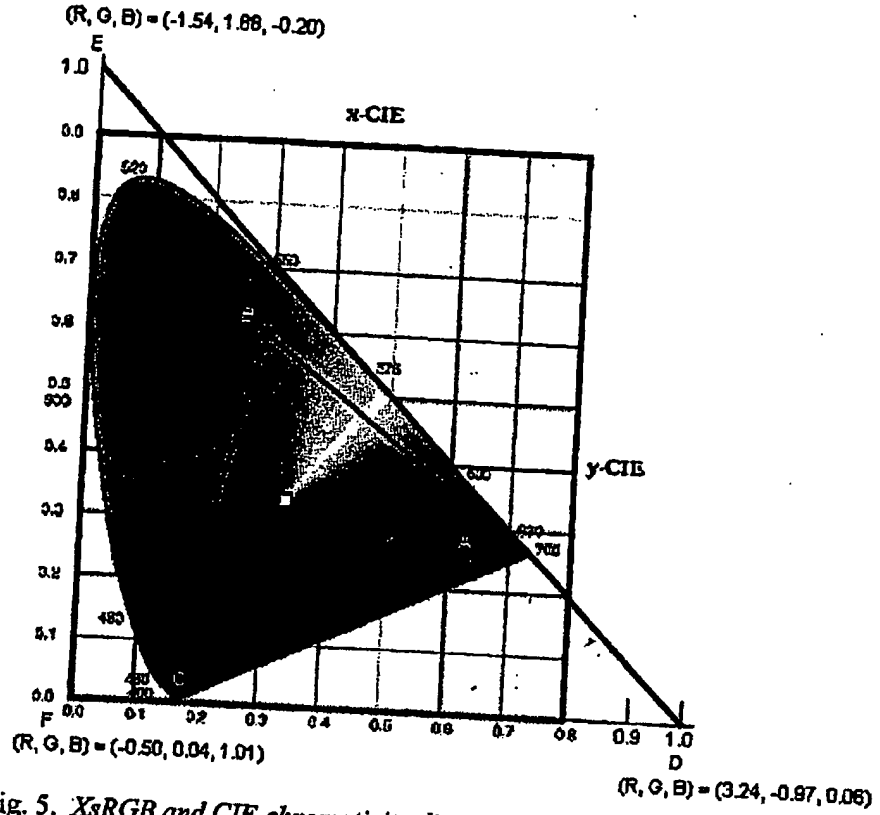

Fig. 5. *XsRGB and CIE chromaticity diagram.*

At the points D, E, and F [where $(x, y) = (1.0, 0.0)$, (0.0, 1.0), and (0.0, 0.0), respectively] in Fig. 4, the normalized XsRGB values are (3.24, –0.97, 0.06), (–1.54, 1.88, –0.20), and (–0.50, 0.04, 1.01), respectively. Also the scaled values of XsRGB give the same points in the xy-chromaticity diagram. Hence, if the components of XsRGB has larger than above range, XsRGB will cover the whole CIE chromaticity and beyond.

When we encode XsRGB in 16 bit in each component, we allow each component go from – 4 to + 4. This range is larger than the chromaticity diagram can cover. Hence we do not have any problem in gamut clipping in our 16 bit each format of XsRGB. When we use 16 bit each version of XsRGB, we must use at least 1 bit for a sign and 2 bits for the integer part. Hence, we have 13 bits for the precisions. We multiply the normalized XsRGB by 8192 ($= 2^{13}$). The 16 bit definition of RGB components of XsRGB is given by

$$\begin{pmatrix} R_{16} \\ G_{16} \\ B_{16} \end{pmatrix} = 8192 \times \begin{pmatrix} R_0 \\ G_0 \\ B_0 \end{pmatrix} \tag{9}$$

Equation (9) is much simpler than the definition of 8 bit sRGB since no gamma corrections are involved. This applies for both cases of D65 and D50.

In sRGB spec, the relation between each 8 bit component and the normalized color component (R_0, G_0, and B_0) are defined. When the value of a sRGB component is small (equal or less than 10), it is related to the normalized component linearly. When it is larger, it is expressed as power of 2.4. The overall power value is regarded as 2.2 which is the value we usually see when sRGB is discussed. For a further details of sRGB, see HP's sRGB Web site (www.srgb.com). For the stardard RGB (8 bit each component) for D50, we used the same gamma correction as sRGB. Hence the following relations apply to conversions to and from XsRGB in both D65 and D50. In the following argument, the word "sRGB" is also used in D50 for the corresponding stardard 8 bit RGB of D50.

The conversion from 16 bit XsRGB to 8 bit sRGB are as follows. Let C_{16} and C_8 denote one of components in 16 bit XsRGB and 8 bit sRGB, respectively. Their relationships are

$$
\begin{aligned}
&C_0 \equiv C_{16}/8192 \qquad \text{(This corresponds to the normalized linear XsRGB)} \\
&C_8 = 0 \qquad\qquad \text{for } C_{16} < 0 \\
&C_8 = 12.92 \times C_0 \times 255 \qquad \text{for } 0 \le C_0 < 0.00304 \ (0 \le C_{16} \le 24) \\
&C_8 = (1.055 \times C_0^{(1.0/2.4)} - 0.055) \times 255 \qquad \text{for } 0.00304 \le C_0 < 1 \ (25 \le C_{16} < 8192) \\
&C_8 = 255 \qquad\qquad \text{for } C_0 \ge 1 \ (C_{16} \ge 8192)
\end{aligned}
\tag{10}
$$

We have clipped blow 0 and above 8192 of 16 bit XsRGB when converting to 8 bit sRGB. This is our default and the easiest way. The color device venders can modify the clipping routines to produce the optimal 8 bit sRGB.

The reverse relation ships are:

$$
\begin{aligned}
&C_{16} = 2.4865 C_8 \qquad\qquad \text{for } 0 \le C_8 \le 10 \\
&C_{16} = 8192 \times [(C_8 + 14.025)/269.025]^{2.4} \qquad \text{for } 11 \le C_8 \le 255
\end{aligned}
\tag{11}
$$

Our extension of sRGB has the following advantages. The blending operations with alpha channel can be directly applied to XsRGB since XsRGB is linear. The color device venders can easily create XsRGB profile from their CIE XYZ profiles. When XsRGB is used for color reference, there is no need to rotate color components to display in 8 bit (each component) sRGB device if the white points of the device and XsRGB are the same. If those white points are different, we have the appearance color matches automatically. Only the gamma correction described in Eq. (10) is necessary to convert to 8 bit sRGB.

The CMM profiles are usually stored in CIE XYZ of D50. If the profile is stored in XsRGB of D50, we are going to have automatic appearance color matches without any further color conversions. The scanned images can be stored in XsRGB format without loosing the bit depths since most scanners produce data in not more than 12 bit in each color component.

VII. Definition of XsARGB and the Interpretation of Alpha Value

For XsARGB, we add the additional 16 bit component. It is used as an alpha channel. Let us first introduce the normalized alpha channel A_0. The values 0 and 1 of A_0 are regarded as transparent and opaque, respectively. The four components (A_0, R_0, G_0, B_0) constitutes one color value. Multiplying 8192 to each component we obtain XsARGB component $(A_{16}, R_{16}, G_{16}, B_{16})$. We called it the non-premultiplied XsARGB and call (A_0, R_0, G_0, B_0) the normalized non-premultiplied XsARGB. The discussion of premultiplied colors are discussed in Appendix A.

When we process blending operations, it is more efficient to use RGB values which are multiplied by the alpha value. We call the four components (A_0, R_0', G_0', B_0'), where $R_0' = A_0 R_0$, $G_0' = A_0 G_0$, and $B_0' = A_0 B_0$, the normalized premultiplied XsRGB. Multiplying 8192 to each component, we obtain XsARGB component $(A_{16}, R_{16}', G_{16}', B_{16}')$. We call this the premultiplied XsARGB.

We have allowed each color component to go beyond 1 and go below 0. How about the alpha value A_0? Can it go beyond 1 or go below 0? We can consider the meaning of alpha in the following way. When we overlay a source image, S, to the distination image, D, we obtain the resultant image, D′, as

$$d' = \alpha s + (1-\alpha)d \qquad (12)$$

where s, d, and d' are one of the normalized color components of the image S, D, and D′ at the corresponding pixels, respectively, and α is the alpha value of the source image S at the considering pixel. When $\alpha = 0$, the resultant image remains the same as the destination image. This case is called transparent. When $\alpha = 1$, the resultant image is the same as the source image. This case is called opaque. When α is between 0 and 1, the resultant image is the mixed image between the source and destination images. Usually α is translucency parameter ranging from transparent (= 0) to opaque (= 1). However, if you look at Eq. (12), it is nothing but the interpolation equation. Hence, when $\alpha < 0$ or $\alpha > 1$, Eq (12) is very well defined and it is extrapolating the source and destination images. Figure 6 shows the alpha value as interpolation/extrapolation parameter. We can allow α to be smaller than 0 or larger than 1. We can call $\alpha < 0$ "super transparent" and $\alpha > 1$ "super opaque."

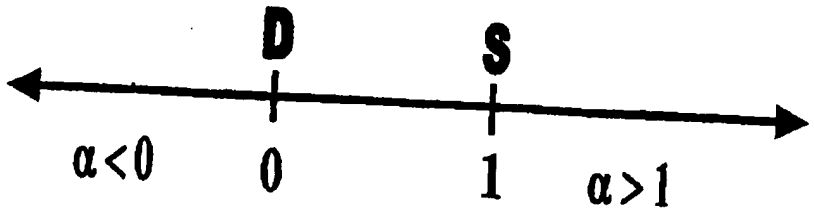

Fig. 6. *Alpha value regarded as interpolation/extrapolation parameter.*

VIII. Data Structure and Image Format

47-32	31-16	15-0
Red	Green	Blue

Table 1. *RGB48 (a XsRGB color in memory)*

When a XsRGB (16bit each component) color is put into a memory, 0-15 bits are the blue component, 16-31 bits are the green component, and 32-47 bits are the red component. Each component is a signed 16 bit integer. The value 8192 is interpreted as 1.0. When it is saved into a file, Intel's Little Endian convention put the first two bytes for the blue component, the next two bytes for the green component, and the supsequent two bytes for the red component. We call this color format RGB48.

63-48	47-32	31-16	15-0
Alpha	Red	Green	Blue

Table 2. *ARGB64 (a XsARGB color in memory)*

Either premultiplied or non-premultiplied, the data structure of XsARGB (16 bit each component) remains the same. When a XsARGB (16bit each component) color is put into a memory, 0-15 bits are the blue component, 16-31 bits are the green component, 32-47 bits are the red component, and 48-63 bits are the alpha component. Each component is a signed 16 bit integer. The value 8192 is interpreted as 1.0. When it is saved into a file, Intel's Little Endian convention put the first two bytes for the blue component, the next two bytes for the green component, and the supsequent two bytes for the red component, and the last two bytes for the alpha component. We call this color format ARGB64.

When the data is stored in linearly, even our 16 bit scale may not be sufficient to store the detailed shades. When more details are required for the certain range of each component, we can assign special range in each component. When each component is normalized, our format can allow each component to vary from -4 to $+4$. However, most values lie within 0 to 1. We can assign special ranges in -4 to $+3$, -3 to -2, 2 to 3, and 3 to 4 to have different scales from the default XsRGB and XsARGB. A user can specify those

ranges and define it in the image header. Hence XsRGB and XsARGB can contain multi-color resolution data.

We can define the variation of color format for lower bit depths. The possible formats are as follows:

32 bit XsRGB – 10 bit in each component (8 bit decimal part) and 2 bit extra.
40 bit XsRGB – 13 bit each component (11 bit decimal part) and 1 bit extra.
40 bit XsARGB – 10 bit each component (8 bit decimal part).
48 bit XsARGB – 12 bit each component (10 bit decimal part).

In order to have a good quality image, we recommend at least to use 40 bit XsRGB and 48 bit XsARGB or higher.

The image format associated with RGB48 and ARGB64 need to be worked out. In this format, 3x4 matrix can be specified as the transform from XsRGB.

IX. Image Filters and Compressions

In addition to the higher accuracies RGB48 and ARGB64 have the extra ranges below 0 and beyond 1 (in terms of the normalized values). This gives significant advantages in image filtering. Suppose we apply a series of filters, f_1, f_2, K , f_n. In each filter, the component value may go below 0 or go above 1. If we truncate the color value in each filter, we may not get the correct result at the end. For an example,

$$\begin{cases} f_1(x) = x - 0.5 \\ f_2(x) = x + 0.5 \end{cases} \tag{13}$$

The correct result of $f_2 \circ f_1(x) \equiv f_2(f_1(x))$ should be x itself. However, if we truncate f_1 between 0 and 1, the result of $f_2 \circ f_1(x)$ becomes 0.5 for all $x < 0.5$.

The existing image format forces the resultant image to be clipped in each filter. Hence, the final result is likely to be incorrect. The will cause the artifacts. With our invention of excess range in the color space, the intermediate results are not clipped. Only the final result is clipped to the range of the output device. Another clipping example is the cubic interpolation. It simulates the sync function and the interpolated value can be negative. Our RGB48 and ARGB64 can store those negative values without clipping.

In general, we want to make the intermediate space as large as possible up to complex numbers. Quantum mechanics uses complex numbers to calculate the wave function. However, the final observable result is expressed as the absolute value of the wave

function as the probability. Unless we keep the complex numbers in the intermediate state, we cannot obtain the correct results. Our RGB48 and ARGB64 are aimed at this direction so that the intermediate states can have very wide range. The general Fourier transform produces the complex numbers. However, we usually use cosine or sine transform that always produce the real numbers. Our RGB48 or ARGB64 format can be used to store the convoluted coefficients of sine or cosine transform which require the signed numbers.

In general, we can store n-dimensional variables $\xi_1, \xi_2, \mathrm{K}, \xi_n$ in the consequetive slots in the RGB48 and ARGB64. Then we can apply the general transform for each variable.

$$F(\omega_1, \mathrm{K}, \omega_n) = \int d\xi_1 \mathrm{K} \int d\xi_n g(\omega_1, \mathrm{K}, \omega_n; \xi_1, \mathrm{K}, \xi_n) f(\xi_1, \mathrm{K}, \xi_n) \tag{14a}$$

If there are inverse transforms, the inverse transformation can be writted by

$$f(\xi_1, \mathrm{K}, \xi_n) = \int d\omega_1 \mathrm{K} \int d\omega_n G(\xi_1, \mathrm{K}, \xi_n; \omega_1, \mathrm{K}, \omega_n) F(\omega_1, \mathrm{K}, \omega_n) \tag{14b}$$

In both cases, the integral can be replaced with the summation, Σ, if the considering variable is discrete. The examples of those transforms are Fouriér transform, (discrete) Cosine and Sine transforms, Laplace transforms, etc. Also Box filters and other filters are other examples.

When the variable is a complex number, we can save it in two component, one for the real part and the other for the imaginary part. We can use the above equations in the complex number and save it in our RGB48 and ARGB64 images as well as in their variation forms.

JPEG compression uses DCT (Discrete Cosine Transform), quantization, and Huffman encoding. In the lossless mode, either DCT or quatization is used. Since all of the above algorithms are well defined in negative numbers, we can use JPEG compression algorithms to compress RGB48 and ARGB64. Since those image formats are not defined in JPEG itself, we need to modify encoder and decoder to be able to handle them. For further details of JPEG compression, see Pennebaker and Mitchell (1993).

When XsRGB or XsARGB contains the multi-color resolution as I described in section VIII, we must compress image separately in usual area and higher resolution areas. We can use Region to describe the area and compress the image in each region for lossy case. For loss-less case, we do not need the special treatment for the multi-color resolution images.

Appendix A: Premultiplied Colors

When we lay an image B on top of an image A, the resultant image P has the pixel value

$$p = \beta b + (1-\beta)a, \tag{A1}$$

where a and b are a color component of the images A and B at a certain pixel and β is the alpha channel of the image B at that pixel. We can overlay an image C on top of the image P to create another image. We want to create the overlay formula so that the final result does not depend on the order (associativity).

$$C \oplus (B \oplus A) = (C \oplus B) \oplus A. \tag{A2}$$

Let D denote the image created by C and B and one of its color components and alpha value are d and δ, respectively. The equation (9) for each color component can be written as

$$\gamma c + (1-\gamma)[\beta b + (1-\beta)a] = \delta d + (1-\delta)a, \tag{A3}$$

where c and γ are one of color components and alpha value of the image C at the pixel of interest. Comparing the coefficients of a, the alpha value of the composited image D must be

$$\delta = \beta + \gamma - \gamma\beta. \tag{A4}$$

Comparing the rest of the equation (A3), the value of the color component, d, multiplied by its alpha value, δ, are given by

$$\delta d = \gamma c + (1-\gamma)\beta b. \tag{A5}$$

Notice that in Eq. (A5) the color components are always multiplied by their alpha values. Hence, it is efficient to work with color components that are already multiplied by their alpha values. Those colors are called premultiplied colors. For further details, see Blinn (1998).

Reference

Foley, et. al, 1990, "Computer Graphics: Princeples and Practice, 2n Ed.", Addison-Wesley.

Starkweather, "Colorspace Interchange Using sRGB", http://www.srgb.com/sRGBColorSpacePaper.pdf

IEC. 1998, "Default RGB color space – sRGB", http://www.srgb.com/sRGBstandard.pdf

Microsoft, "Color Management in Microsoft Windows Operating Systems", http://www.microsoft.com/windows/platform/icmwp.htm

Blinn, J. 1998, "Jim Blinn's Corner, Dirty Pixels", Chapter 16., Morgan Kaufman.

Pennebaker, W. B. and Mitchell, J. L. 1993, "JPEG Still Image Data Compression Standard", Van Nostrand Reinhold.